CREATION OF THE SELECTION LIST FOR THE EXPERIMENT SCHEDULING PROGRAM (ESP)

Prepared for:

G. C. Marshall Space Flight Center
Systems Analysis and Integration Laboratory
Marshall Space Flight Center, Alabama, 35812

by:

Dr. Bryan L. Deuermeyer
Dr. Robert E. Shannon
Alvin J. Underbrink, Jr.

Industrial Engineering Division
Texas Engineering Experiment Station
Texas A&M University
College Station, TX, 77843

FINAL REPORT

Contract No. NAS8-35972

May 15, 1986

CREATION OF THE SELECTION LIST FOR THE EXPERIMENT SCHEDULING PROGRAM (ESP)

Prepared for:

G. C. Marshall Space Flight Center
Systems Analysis and Integration Laboratory
Marshall Space Flight Center, Alabama, 35812

by:

Dr. Bryan L. Deuermeyer
Dr. Robert E. Shannon
Alvin J. Underbrink, Jr.

Industrial Engineering Division
Texas Engineering Experiment Station
Texas A&M University
College Station, TX, 77843

FINAL REPORT

Contract No. NAS8-35972

May 15, 1986

ABSTRACT

This report summarizes the efforts to develop a new procedure to construct selection groups to augment the Experiment Scheduling Program (ESP). Included in this report is a User's Guide and a sample scenario to guide the use of the software system that implements the procedures developed under this contract.

Table of Contents

1.0 PROJECT SUMMARY

The primary goal of this project was to develop a procedure for creating the selection list that serves as the input for the Experiment Scheduling Program (ESP). Currently, ESP removes experiments from the selection list using either the top experiment, or a randomly selected performance. The supporting software for ESP does not analyze the mission data to construct a selection list — this task is left up to the engineers as part of the modeling activity. The one major restriction imposed on the research team was that the internal operation of ESP could not be modified. Thus, the research effort focused upon methods for establishing a selection list that would work in harmony with ESP and yet would create good timeline schedules in less time and effort than the present manual approach. A secondary goal of the project was to create a new optimization procedure for solving a subset of the experiment scheduling problem. This procedure could eventually be factored into ESP to enhance its performance. The tasks performed to date are summarized in the last section of this report. Appendix 2 provides a short glossary of terms used in this report.

The culmination of this project is a software package that implements the hybrid approach motivated and developed in this report. Appendix 5 provides a User's Guide to the software. While this package is not integrated into the menu for ESP, it does access and modify the ESP data structures and files through service routines provided by the Marshall Space Flight Center. The envisioned usage is for the engineer to create selection groups that are subsequently used by ESP for creating a time line.

The overall methodology developed and implemented during this project explicitly categorizes experiments into three broad groups: (1) critical experiments and those experiments that serve as predecessors for other experiments; (2) experiments with mandatory and necessary concurrence; and (3) all other experiments. The most sophisticated and automatic procedures apply only to group (1), which is also the most difficult to hand schedule due to the sequencing requirements. The least automated but most flexible procedures, based upon clustering and ranking, apply to group (3). Methods for group (2) are similar to those applied to group (3), except that random sequencing is recommended

for these experiments. Therefore, if an experiment file consists of experiments which predominately fall into groups (1) and (2), our methodology will produce very high quality timelines with virtually no effort on the part of the engineer. On the other hand, if the experiment file has little sequencing or concurrence requirements, much more of the burden is shifted to the engineer.

2.0 Background

Our literature review consists of approximately 45 papers, taken from resource scheduling, project scheduling and machine shop scheduling (Appendix 1 provides a detailed review of this literature). After reviewing this literature, it quickly became apparent that the experiment scheduling problem is more general and more complicated than the problems reported. It is important to identify those features of the experiment scheduling problem that give it such special character:

1. Each experiment to be scheduled may have multiple performances;
2. Each performance has a series of steps that must be accomplished in order;
3. Some steps have a minimum, as well as a maximum step duration, with the minimum being mandatory. Ideally as much time as possible should be given to steps;
4. Mandatory, necessary and desirable concurrence must be considered;
5. Some experiments have sequencing requirements;
6. Some experiments/steps have target requirements and these translate to time window requirements;
7. Both depletable and non-depletable resources must be considered;
8. Some performances of an experiment have a startup step and/or a shutdown step – other performances do not have these steps;
9. Delays may be required between steps and/or performances;
10. Resources needed by one experiment may be created by other experiments;
11. The mission length is fixed, which implies that all experiments that are to be performed must be scheduled within the mission window. In scheduling vernacular, the makespan is fixed.

These features make the experiment scheduling problem far more difficult than the job shop problems studied in the scheduling literature. Actually, the problem addressed by ESP corresponds more closely with real-world production scheduling problems, not with the more simplified situations modeled in the literature. However, the time complexity of job shop problems, which can be viewed as subproblems of the experiment scheduling problem, suggests that no optimum seeking approach will be computationally feasible. Heuristics offer the only hope to approaching the problem; the experiment scheduling problem inherits this unfortunate property. Therefore, heuristics based upon ranking principles appears to be the only practical way to approach the scheduling problem.

The initial phase of the project identified the following three approaches for creating the selection list:

1. Utilize and develop ranking procedures.
2. Utilize techniques from production and computer system scheduling, as well as resource scheduling.
3. Develop an optimization model and heuristics.

After some work on the experiment scheduling problem it became apparent that this problem has several features that distinguish it from both production and computer system scheduling problems as well as resource scheduling problems found in the traditional literature. Also, it was determined that simple one-step ranking procedures will not perform to the desired level of performance. Consequently, an alternative approach that augments the simple ranking method with clustering was selected. Cluster analysis allows the engineer to explicitly determine the experiments that have similar scheduling requirements to aid in the construction of a selection list. Once clusters have been formed, they are ranked and the ordered clusters of experiments are then placed into one or more selection groups.

3.0 Ranking Procedures

Almost any numerical ranking procedure can be expressed in the following manner. Suppose there are N factors of interest, that factor j is given a numerical weight α_j, and that u_{ij} is the score given to factor j of experiment i. Then the ranking score for experiment i is calculated by the expression

$$r_i = \sum_{j=1}^{N} \alpha_j u_{ij}. \qquad (1)$$

The most straight forward method for setting up the selection list is to sort the experiments into decreasing order of ranking score. At this point it is worthwhile to provide some of the background behind ranking procedures and their application to scheduling problems. In the most simplistic scheduling problem, a single resource is to be used to perform one task on each of a collection of independent jobs. For many of the basic performance measures (e.g. mean flow time), the best schedules can be obtained by simply sorting the processing times, due-dates, or ratios of processing times to costs. For other criteria, such as mean tardiness, an optimal schedule cannot be identified in a straight forward manner. As a result, heuristic procedures have gained popularity, both for pragmatic as well as performance reasons. Most heuristic procedures are called dispatching rules since the next task to perform at a processor (resource) is determined by looking at the collection of tasks currently present, and selecting one based on a single attribute. For example, one could select the task having the shortest processing time, the least amount of work remaining, the largest critical ratio, etc. Once again, dispatching rules amount to sorting tasks based upon a single criterion.

Many real scheduling problems do not have a single dominant criteria, and so no one dispatching rule is capable of providing the desired performance. Attempts to circumvent this shortcoming by assigning scores to each of the performance criteria for each job/task and then computing a weighted average of the criteria have been proposed. Despite the logical reasoning behind this approach, there has not been wide success with it. There are several technical reasons for the lack of general success.

First, since performance measures are often conflicting, a schedule that works well for one measure may be poor for others. Therefore, simply weighting the measures will not necessarily produce a schedule that does even reasonably well for any measure. Desired performance is only achieved by careful selection of the weights; and this frequently reduces

to ranking the importance of the measures and adjusting the weights to produce the desired result.

Second, dispatching rules, and hence the ranking scores are typically computed at each machine or operation. It is extremely difficult to estimate the down stream effects of all of these local decisions. For example, if work is selected at machine 2 based upon its current and future work content, it may well be that a task is selected for processing because the queue at the next work station for this job may currently be empty. However, several work stations may make the same decision for the same reason. A short time later a serious bottleneck may result. To date, such simple approaches have not worked due to the lack of ability to accurately predict overall shop loadings and to make this information available to individual work stations. Hence, the real problem is one of properly addressing resource availabilities and factoring this information into dispatching rules.

Third, as mentioned above, aggregate scoring techniques are highly dependent upon the weights. It is usually not possible to settle on values for the weights that work uniformly well for all job sets. Some degree of experimentation is necessary to yield the desired performance.

Fourth, it is possible for two jobs to produce the same score due to entirely different reasons. It may make sense to handle these jobs differently even though their scores are equivalent.

4.0 Ranking Procedures and ESP

There have been two suggested ranking procedures offered for the experiment scheduling problem. The first, called *resource scoring* (RS) , assigns a value to each resource, number of performances, etc. and then computes a weighted average of the form given by equation (1). Weights here reflect the desired importance associated with the factors. The second, called ***difficulty scoring*** (DS), assigns a value to factors that should impact the schedulability of the experiment. Weights are applied to each of these factors, again reflecting importance of the factors. At least conceptually, the DS method is superior to the RS method in that the focus is directly upon schedulability.

It turns out, however, that neither of these procedures can be expected to produce good schedules all of the time. One of the most important reasons that ranking procedures, such as the ones above, do not work well is that the weights are not usually dependent upon the mission data. Thus, weights that work for one mission will probably not work on another mission. There is no way to avoid experimentation to fine-tune the weights, and it does not seem possible to provide general guidelines to govern this process. In the end, one schedules by experimenting and learning what is important and what works on a case by case basis. Of course, this defeats the reason for using a scoring method. A second reason is that the time dependent availability of resources, sequencing requirements, and other aspects of the problem require more than one rule to get good schedules.

Our research (Appendix 3 provides a synopsis of representative experimental runs throughout the research project) has identified an alternative approach that appears to work well with ESP; it is an approach that is generalizable and modifiable, and experience from one mission to another can be factored into the approach. The early work on this project attempted to access the applicability of these two approaches by trying to determine how various measures of difficulty and resource utilization impact schedule performance. The justification for this approach was that the ESP program and its logic were not to be changed — only the selection group construction was to be addressed. In the beginning, this seemed to be severe restriction; however, as the work progressed, two insights were gained. First, ESP does a remarkably good job of scheduling given the selection list and what ESP does is remarkably complicated. Second, much of the theory or reasoning behind the scoring methods miss crucial aspects of the problem. Given that only selection grouping was open to study, the DS approach (or more sophisticated variants to it) at first seemed very reasonable. However, carefully constructed test groups revealed that no single ranking procedure will work in general.

Let us take a closer look at what ranking procedures do and then see if a stronger and more versatile approach is suggested. Numerical ranking methods assign a score to each experiment and then the experiments are sequenced by value, so that the most difficult to schedule experiments are given to ESP first. If you look at a selection list constructed in this manner what you will find are groups of experiments with similar ranking scores —

like the results of an examination. As with examinations, you can see the A's, B's, etc., and it is these that are the groupings. In grading examinations, this approach is acceptable because the purpose of the examination is to measure a student's mastery of the subject; the groups identify classes of performance. It is important to recognize that the global affect of a numerical ranking procedure is that it does identify classes of experiments just as in the examination case. But, what are the implications of these classes?

First, the difficulty factors that are used in the weighting are not homogeneous — they each impact timeline creation in a different way. As mentioned above, two experiments may have the same score but achieve this score for different reasons. More generally, experiments in the same scoring class (i.e. the A's, B's, etc.) may have distinctly different impacts on scheduling.

Second, different numerical weights will create different experiment classes and probably different scheduling performance. When two missions differ significantly, it may be necessary to conduct an extensive trial and error process to find the weights that produce the desired performance.

Third, little overall insight can be gained from the trial and error approach because the resulting classes and rankings are not explicitly available to the engineer. What is actually needed is a method that forms the classes directly by similarity of categories. What is meant here is that two experiments are in the same class if their vectors of categories (by category we mean lists of characteristics such as those used in numerical ranking) are more similar to each other than to experiments not in their class. Once such classes, called clusters, have been created, the engineer would rank the clusters.

This discussion hopefully points out that numerical ranking methods do in fact produce ordered groups of experiments, but that membership in these groups is not well-founded and the factors that govern the membership are inaccessible to the engineer. We suggest using an explicit cluster analysis before ranking to allow the engineer more control over the creation of the selection list.

Before moving on, it is important to point out that no ranking procedure can successfully address sequencing requirements. Therefore, we present an algorithm that directly handles such requirements. The disscussion appears in sections 5 and 6.

Section 7 describes cluster analysis and how it can be used to provide an alternative to the simple numerical ranking procedures discussed above.

5.0 A Hybrid Approach

This research has identified the follwing three classes of experiments and presents techniques to address each class; the TAMU software package represents an implementation of these techniques.

1. Experiments that are critical to other experiments because of sequencing requirements;
2. Experiments that have either mandatory or necessary concurrence requirements; and
3. All other experiments.

This partition of the mission file is in recognition of the distinct characteristics that these classes have in terms of their scheduling requirements. Our experience with SPL.MOD indicates that the quality of the overall timeline depends very heavily upon the quality of the sequencing of the experiments in class 1; furthermore, this is the most difficult set of experiments to hand schedule due to their down-stream impact. Class three consists of such a wide variety of experiments that we are forced to consider some type of ranking procedure, but the approach we offer is based upon clustering and interactive use by the engineer as apposed to the less direct numerical scoring techniques mentioned in the previous two sections. We offer a clustering technique and interactive software support to sequence class three.

The approach will be to create a minimum of three ESP selection groups, one for each of the above classes, and then these three selection classes should be processed in order by ESP. In the following sections of this report, we describe the methodologies that should be used to partition the mission file into these three classes and to sequence the experiments within each class.

Function [1] of the TAMU system creates the three classes and the associated three (unsequenced) ESP selection groups, and prepares the necessary data structures to perform the subsequent steps. Function [2] the sequences the experiments in the first selection group, those experiments that must be completed as prerequisites for other experiments. Functions [3] - [5] facilitate cluster analysis and the construction of the third class of

selection groups (for those experiments not being predecessors to other experiments nor having concurrency requirements). Finally, function [6] prepares the selection groups for access by ESP.

What remains to discuss is class 2, the set of mandatory and necessary concurrence experiments. We can summarize our experience with SPL.MOD for class two as follows:

1. While we found that clustering offers a lot of assistance for this set of experiment, we found that using ESP's random order sequencing rule to be very effective here. The reason is quite simple: this rule schedules on a performance by performance basis. Since the number of experiments is not large, ESP can schedule these experiments quite quickly.
2. We did find that it makes sense to split this selection group if there are experiments with a large number of requested performances; the second sub-selection group would not be sequenced using the random order rule, and we recommend that this second group be placed after the selection groups obtained from class three.

6.0 Handling Sequencing Requirement

The purpose of this section is to describe the approach for handling sequencing requirements. This is a difficult problem and finding a good solution is both hard and critical. Typically, a mission file contains a relatively small number of experiments that must be performed as prerequisites for most of the remaining experiments. Unfortunately, these critical experiments fall into independent classes such as depicted in Figure 1. Notice that there are two "trees" one rooted at experiment number 26 and the other at experiment 133. Also notice that in terms of sequencing requirements, these two trees are independent. The crux of the algorithm discussed in this section is a heuristic for ordering and merging these trees.is a difficult problem because a mission file.

The first step is to search through the model file and assess the overall precedences that must be followed. A convenient method to accomplish this is to determine a spanning forest, consisting of the collection of trees describing the sequencing requirements. In general, there will either be an empty forest, which occurs if there are no sequencing requirements, or there will be several trees in the forest. Figure 1 provides a partial picture

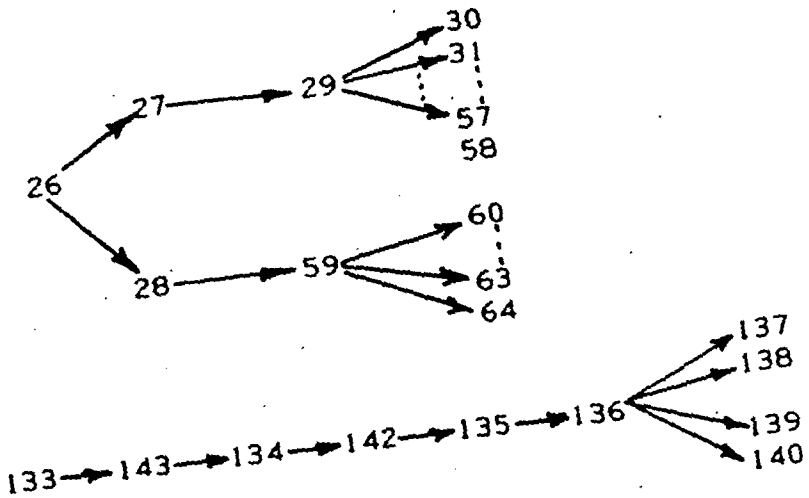

30
31
57
58
29
27
26
28
59
60
63
64
137
138
139
140
136
135
142
134
143
133

Figure 1

of such a spanning forest. Therefore, the first pass of the algorithm (TAMU menu function [2], and implemented in subroutine WEISS) must construct an internal representation of this forest. In addition, the algorithm must identify those experiments that are leaves of the spanning forest; these experiments are excluded from selection group 1.

For example, in the spanning forest rooted at experiment 133, notice that experiments 137-139, and 140 cannot be scheduled unless 133, 143, 142, 135 and 136 have already been scheduled. For this reason, these last five experiments are assigned to class 1. The general rule that we have developed is that any experiment having at least one successor is assigned to class 1; all other experiments are assigned to class 2 if they have either mandatory or necessary concurrence and to class 3 otherwise. There are two exceptions: any experiment having mandatory or necessary concurrence with experiments in class 1 are also assigned to class 1, and other critical experiments may also be assigned to class 1 to insure that they have an opportunity to be scheduled. Experiments in this last group are judged to be critical by the engineer based upon considerations beyond the scope of ESP.

This first pass accomplishes its task by first setting up an adjacency matrix (a matrix which has a row for each experiment and each row lists the experiments that follow the designated experiment). A depth-first traversal determines the leaves of the tree and sets up the data structures for the second phase of the algorithm, which is to order the experiments that fall into class 1.

Phase 2 of the algorithm is based upon a heuristic developed by Weiss (1981). The idea is to view the current forest as a partially ordered collection of experiments. Edges are added to the forest during successive steps of the algorithm until the forest becomes a chain; that is, exactly one tree remains, and this tree is completely ordered.

The algorithm selects an edge based upon a pre-coded function $C(i, j)$, which specifies a "cost" of placing experiment i before experiment j, for experiments i and j that are not already ordered into a chain. The experiment with the smallest such function value is selected and the corresponding edge is inserted into the forest. The algorithm next updates the transitive closure (i.e. also adds any edges implied by the current forest as a consequence of adding the new edge). This process is repeated until there is a total of

$\frac{n(n+1)}{2}$ (n is the number of experiments in class 1) edges in the forest; at this point the order experiment order is established.

Currently, experiments in class 1 are assigned to selection group 1, and are sequenced by experiment number. A further refinement makes use of the performance window size and location to prioritize of these experiments. The hooks are present to allow the engineer to code in a more general function, C, than we have used; this is documented in the code for WEISS.FOR. ESP already schedules all of the experiments that we have placed into selection group 1 except those experiments that have to be hand scheduled or that have errors in their model. While this performance may depend upon the specific characteristics of the test model file that we used, we do expect a high quality of performance on other missions as well.

7.0 Cluster Analysis

The role of clustering in the scheduling problem is to provide the engineer with a standard basis for grouping and scheduling those experiments that do not have either sequencing requirements nor concurrency requirements. The approach discussed in this section requires the engineer to write his own program to categorize the characteristics of the experiments based upon those factors thought to be important. The philosophy is to make it possible for the engineer to make maximum use of the knowledge that he or she has concerning both the experiment data and using ESP. Successful application of the method may require some trial and error when the engineer encounters a model file (experiment file) that is a radical departure from previous cases. For example, if past situations have included a lot of sequencing requirements, concurrence and targets and a new case contains none of these, then experimentation is necessary to recalibrate the method (i.e. determine which factors are critical to scheduling this new mission). Since our overall hybrid scheduling procedure works best when sequencing requirements are extensive, it is recommended that modeling be conducted with this in mind. Going a step further, more can be accomplished from our system if more attention to scheduling issues is paid during modeling.

Factor	Definition
Perf1	Number of performances ≤ 5
Perf2	Number of performances > 5 and ≤ 10
Perf3	Number of performances > 10
Wsmall	Window length ≤ 72 hours
Wmedium	Window length > 72 and ≤ 144
Wlarge	Window length > 144 and ≤ 216
Whuge	Window length > 216
Wmult	More than one performance window exists
TgtI	N3F05, OPEN ATT, SEPAC, WAM-8 , SEPA1, X3 F
TgtII	IVL60-80, LAT, MAG
TgtIII	LMT222-2, NOONMDNT, 2QNIGHT, SHADOW, MSHADOW
TgtIV	DAY, NIGHT, SBAND, SUN, TDRS
TgtV	A target is present, but not critical or no target is required

Table 1 — Definitions for Cluster Factor

The clustering approach relies on the premise that a few factors are critical to the ability for ESP to produce good schedules, and that the entire collection of experiments to be scheduled fall into a few (on the order of 10) major groups. Furthermore, there is a reasonable ordering of the groups based simply on what factors are present within the groups. Below is a discussion of how to carry out the cluster analysis and to order the clusters. Appendix 4 details the actual clustering algorithm.

Appendix 7 provides a sample listing of cluster analysis as applied to those experiments in SPL.MOD that do not have either concurrency or sequencing requirements. Three broad categories of factors were included: (1) Number of performances; (2) Window width; and (3) Target requirements. Several levels of each factor were created within each broad category to reflect varying degrees of criticality to scheduling. Table 1 provides the definitions of the factors. Those targets in TgtI were judged to be the most difficult to schedule, while going from TgtII to TgtV are progressively easier to schedule — in terms of targets alone. Similarly, Wsmall is the most difficult to schedule and Whuge is the easiest. Wmult was included because there was a small number of experiments having multiple performance windows.

These choices were made partly due to our experience with SPL.MOD and due to consultation with personnel from Marshall Space Flight Center; the latter specifically in categorizing the targets by criticality of scheduling. This choice produces 13 distinct factors. Now, a preprocessor is required to loop through the model file and score each experiment relative to the 13 factors. If a factor is present, then that factor receives a score of 1, otherwise that factor receives a score of 0. (The preprocessor is accessed from function [3] of the TAMU menu, and is transparent to the user. Appendix 6 discusses this in more detail. However, the program ROUTMOD.FOR must be modified by the user if different factors are desired.) To illustrate the scoring, consider experiment number 157. Factors Perf1 and Whuge receive scores of 1 and all other scores are set to 0. This information is contained in the file PAK.RPT, obtained by selecting function [4] in the TAMU menu and is the result of the first pass of the clustering algorithm.

The first pass of the clustering algorithm is to collect experiments into groups, called packs, if they have exactly the same scoring vectors. Experiments falling into the same pack are equivalent so far as clustering is concerned. The pack report is arranged into three basic groups of columns. The first column lists the pack number, the second two columns list the experiment numbers of the experiments contained in the pack and the third group of columns lists the factors present in each of the experiments in the pack. For example, Pack number 2 consists of experiments 2, 4, 5, 6, 7 and 91, and each of these experiments use factors Perf3, TgtIV and Wsmall . The purpose of the packs is simply to reduce the size of the problem to be considered by the clustering algorithm. Thus, out of the 186 experiments there are only 32 packs. Once again, experiments within a pack are judged to be equivalent (i.e. have the same set of requirements) so far as clustering is concerned.

Before clustering can begin, the system must compute the similarity matrix, which specifies a numerical value called the similarity index for each pair of packs. The larger this value, the more similar the two packs; if there are no common factors, the index is zero. Again, the reason for using packs is to minimize the size of this matrix.

Once the similarity indices have been computed, the clustering algorithm sets out to create the clusters. To do this, the system repetitively takes a pack and compares

Pack No.	Perf1	Wmult	TgtIV	TgtIII
15	*	*		
16	*	*	*	
19	*	*		*

it to previously determined clusters. If this pack is sufficiently similar to one of these clusters, then it is added to the cluster it best fits; otherwise the experiment initiates a new cluster. The actual details of how all of this is done and how the sensitivity of the comparisons are controlled is described in Appendix 4. The outcome of the clustering analysis is documented in the file CLU.RPT, produced by function [4] in the TAMU system.

To explain the cluster report, we consider its two main parts. The first is the three lines at the beginning of the report. The threshold density is 2, representing a moderate sensitivity, 32 packs were considered and 11 clusters were formed. The second part of the report is a listing of each of the 11 clusters. For purpose of discussion, consider cluster number 4.

This cluster contains packs 15, 16 and 19 and hence experiments 66, 73, 80, 67, 72, and 78, in that order. The top line of this section lists Perf1, Wmult, TgtIV and TgtIII, indicating that the experiments in this cluster have these factors. Below this information is the component pack matrix. The columns of this matrix are in one-to-one correspondence with the list of factors found at the top; an asterisk indicates that the factor is present. To facilitate the discussion, consider the following expanded copy of the pack matrix. Notice that all packs (i.e. sets of experiments) in this cluster have both Perf1 and Wmult, but they differ in what targets are required. At first it would appear that an ideal cluster would be one where there is an asterisk in each position of the matrix; however, since this condition is already treated in creating the packs, what is typically observed are clusters similar to those in the table. Notice that as the cluster proceeds downward, the matrix has the appearance of spreading out to the right. This affect is even more pronounced in some clusters. While there are no absolutes, what we want to avoid is several clusters that exhibit pronounced sparsity. The reason is that as packs are added, they are less and less similar to the earlier packs. This behavior is generally caused by using too large a value

for the sensitivity parameter k. Reducing k forces the system to create a larger number, although more similar, of clusters.

What have we accomplished so far? To begin with, we now only have to sequence the 11 clusters, which is easier to handle than the original number of experiments. Furthermore, we only need to keep track of 13 factors, and our ordering will be based on these alone. Let us summarize where we have been and what yet needs to be done.

The clustering process (assuming factor categories have already been established) requires three general steps:

1. Create packs and then clusters.
2. Analyze clusters and determine cluster ordering.
3. Create the selection group(s) (i.e. one or more hoppers labelled 3,4,...,11) for ESP.

We now detail these specific tasks that must be carried out to accomplish the steps mentioned above, and include in this discussion the software support that is available. Appendix 6 provides the complete dialogue of the sample session that created the pack and cluster report provided in Appendix 7.

1. Select item [3] on the TAMU menu - this function automatically creates the packs and clusters for those experiments that do not have either sequencing or concurrency requirements. The system will prompt for a "k" value that selects the clustering sensitivity. A value of 2 generally works best. (Smaller values of k produce more clusters while larger values of k produce fewer clusters.) Once control is passed back to the engineer, select function [4] from the menu to have the system create the cluster report, which can then be viewed by a TYPE CLU.RPT command from operating system level. Once a hard copy of the cluster report is in hand, proceed to the next step.
2. Once the clusters have been formed and the cluster report is available, the next step is to analyze the nature of the clusters and to use this information to sequence the clusters. We have found that the order that the clustering analysis provides within clusters works quite well, so the effort needs to be directed at ordering the individual clusters. The method we had success with orders the clusters by repetitively placing first the cluster having the most highly critical factors. The engineer should do this

Hopper	Selection Group	Cluster Sequence
1	3	1, 7, 9
2	4	11
3	5	6
4	6	2, 3, 5
5	7	4, 8, 10

based upon past experience, knowledge of the mission, or trial and error. The software that supports this task is accessed by selecting function [5].

3. Once the clusters have been ordered, select function [6] from the menu to prepare the experiment file for use by ESP. At this point, work with the TAMU software is complete and it is time to go to ESP. Select function [7] to exit the system.

Step two, the ordering of the clusters deserves further discussion. The dialogue in Appendix 6 provides an example where 5 hoppers are created from these 11 clusters. The table below shows the hoppers created by this step, the selection group number used within ESP, followed by the cluster sequence within the designated hopper.

The ordering was produced by a very simple, yet effective scheme. First, the clusters were arranged in the order such that Wsmall - Wmedium - Wlarge - Whuge - Wmult, since we found that window width was the single most important factor (in SPL.MOD). Wmult was placed last because there is more flexibility here than in the other cases. Next, within this ordering, we put a bias on placing clusters containing TgtI (the most critical targets) first. Hopper 5 contains the clusters where Wmult is present, and these clusters were arranged basically by smallest number of performances first.

The summary of experimental results provided in Appendix 3 is a representative sample of our trial and error runs. We can summarize as follows.

1. Performance window length was most important among factors;
2. It is advantageous to use several hoppers as output of the clustering analysis due to the excellent editing capabilities in ESP.

In conclusion, clustering provides a means to reduce the amount of information needed and the size of the problem that the engineer must consider in order to sequence the experiments that do not have either sequencing nor concurrency requirements. Ultimately the engineer is forced to manually do the sequencing, but this should only involve a small

number of objects, the clusters. In this manner, individual experiments do not have to be dealt with explicitly and the factors and cluster summary information makes it easier for the engineer to work with the most critical aspects of the problem.

8.0 Optimization Approach: Project Scheduling

This section describes a parallel effort which sought to develop an optimization algorithm for sequencing experiments within packs and clusters within hoppers. Suppose that a set of activities related by transitive precedence relations is given. The duration of each activity is assumed to be a known constant. In addition, information regarding the starting and finishing times of each activity such as ready time, due date, and time delay between two consecutive activities may also be available. In this case, constraints must also be incorporated to ensure that these time requirements are met.

Activities require the use of certain resources during their execution. The number of units of each resource type is necessary to perform each activity is assumed to be a known constant as well as the number of units of each resource type available at each time period. Other constraints such as concurrence may also be included, in which case some activities are required to be performed simultaneously. Our approach builds upon the formulation of Balas (1970).

Suppose that there are n pairs of disjunctive arcs in a graph. Let X_k be a variable that indicates the state of disjunctive pair k. Let $\vec{X}$ be the vector of variables representing the states of all disjunctive pairs in the graph. A selection is given by one particular value of $\vec{X}$. Moreover, a selection is said to be complete if, for all disjunctive pairs, exactly one of arcs is active. Having the above definitions, it is now possible to formulate the project scheduling problem. For simplicity it is assumed that there are only precedence and resource constraints and that resource availabilities are constant over time. Other constraints as well as variable resource availabilities are easily incorporated into the model, but at the expense of more complicated notation.

The following discussion will use the so called activity-on-node convention. In order to illustrate the concepts discussed above, an example problem is presented in Figure 2. Additional data is presented in Table 1, where $r_i(j)$ is the usage of resource i by activity

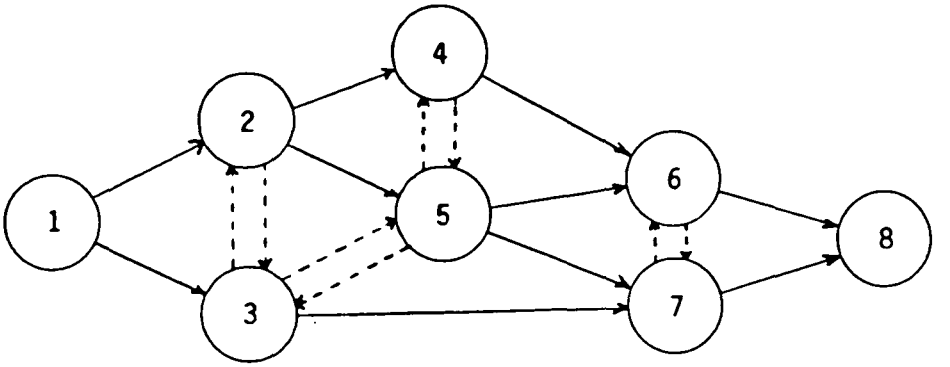


Figure 2

Activity	Duration	$r_1(j)$	$r_2(j)$	$r_3(j)$
1	2	1	1	1
2	5	2	2	1
3	4	1	0	0
4	2	0	2	0
5	1	1	1	1
6	3	0	1	1
7	2	0	0	2
8	0	0	0	0

$R_1=2$, $R_2=2$ and $R_3=2$.

Table 2

j and R_i is the availability of resource i. Suppose, without loss of generality, that there is one initial activity with no predecessors and one end activity with no successors. Suppose also that all other activities have at least one predecessor and one successor. To each arc leaving activity (node) i is associated the duration of activity i. Now, between all pairs of activities which are not precedence related and which consume the same resource create a disjunctive pair of arcs. Again, the parameter associated with each of the arcs is the duration of the activity where the arc originates. For the set of disjunctive pairs so created, a selection is said to be feasible if no resource conflict is present. Considering minimization of project duration as the objective criterion, the project scheduling problem can now be formulated as the problem of finding, among all possible feasible selections, the one which minimizes the critical path in the network. One example of a feasible selection for the problem presented in Figure 2 and Table 2 is given in Figure 3.

A Decomposition Procedure

The idea of a decomposition procedure comes as an effort to develop a more consistent heuristic procedure. The basic idea is to divide the problem into a sequence of subproblems which can then be solved, if possible, optimally. Decisions are made only as they become necessary.

The procedure starts with a solution which provides a lower bound on the total project duration. This solution can be obtained by deleting the resource constraints and computing the earliest start time for each activity by means of a critical path analysis. The next step in the procedure is to scan the time line until a resource infeasibility is found. At this point a subproblem is constructed. The activities in the subproblem are the ones which are involved in any resource conflict at the current point in time. Notice that all activities involved in a resource conflict are not precedence related. The model will depend on the criterion used to solve the subproblems. In this project, minimization of makespan is of interest. For simplicity, it is also assumed that activities which are already initiated at the current time are not considered for removal although the general case where any activity is considered for removal could be considered. A theoretical reason for such an assumption is that, in this case, the solution obtained for the global problem can be shown to be active;

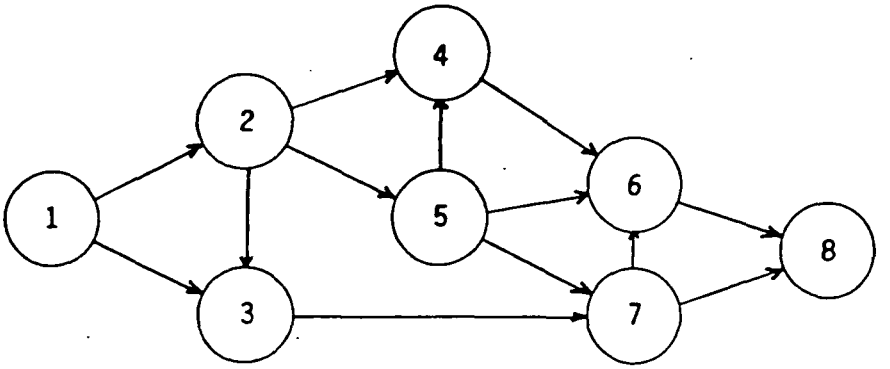


Figure 3

Activity	duration	$r_1(j)$	$r_2(j)$	$r_3(j)$	$r_4(j)$
1	4	6	2	2	6
2	2	5	1	5	1
3	5	4	3	4	3
4	3	1	3	5	5

Table 3

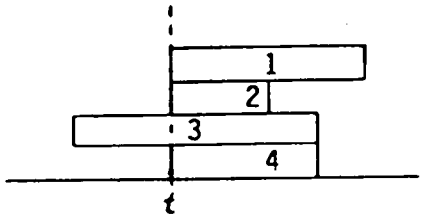


Figure 4

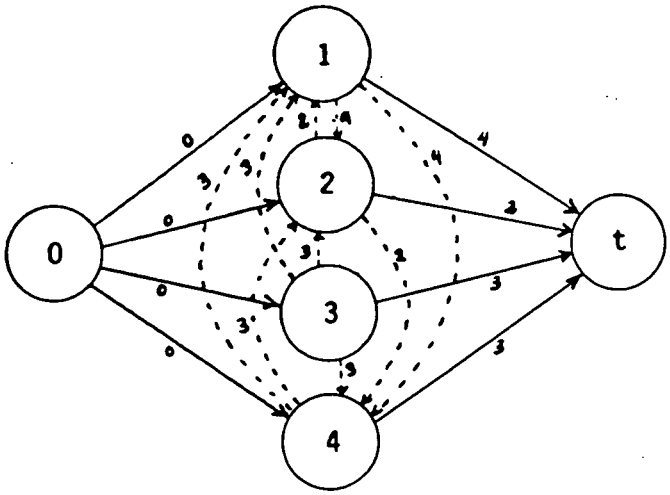


Figure 5

in other words, no left shifting of an activity is possible without increasing the completion time of some other activities. As an example, consider the situation illustrated by Figure 4, where activities are arbitrarily labeled. Also, consider the data presented in Table 3.

Assume that the resource availabilities for the four resources are 10. The model for minimizing makespan for this subproblem is given by the network in Figure 5, where nodes 0 and t are dummy nodes. Notice that, in this example, activity 3 was already started. As a consequence, none of the other activities can precede activity 3. For this reason, only the arcs leaving activity 3 are considered to become active. Notice also that the parameter associated with the arcs leaving activity 3 is equal to the total duration of actvity 3 subtracted by the portion already completed by time t.

In order to find the best feasible selection for each subproblem, a concept which is reported in Bellman, Esogbue and Nabeshima (1982) is utilized. Suppose A is the set of activities in subproblem 1. In the previous example, $A = \{1, 2, 3, 4\}$. Create all subsets of size 2 of set A such that no two activities in the same subset can possibly be scheduled simultaneously. In the example, $A1 = \{1, 2\}$ and $A2 = \{1, 4\}$. Now, create all subsets of size 3 of set A such that the three activities in a given subset can not be scheduled simultaneously and such that the new subset of size 3 does not contain any of the subsets of size 2. In the example, the only subset of size 3 is $A3 = \{2, 3, 4\}$. An interesting property of the new subset so created is that, by creating exactly one precedence relation between any two activities in the new subset, all resource conflicts among those activities in the subset under consideration are resolved. The process is repeated by creating subsets of greater size until no more subsets can be created.

Suppose the total number of subsets of set A so created is m. In the example, m is equal to 3. Notice that for each pair of nodes appearing in the subsets $A1$,..., Am, it corresponds to one disjunctive arc in the model. Now, suppose that exactly one pair of nodes is selected from each of the subsets. By making exactly one of the arcs connecting each pair active, the solution so generated will either be feasible or contain a circuit, in which case the solution is discarded. Moreover, it can be shown that by generating all possible combinations of pair of nodes, each pair belonging to one subset, all possible feasible solutions will be investigated. This fact guarantees an optimal solution can be

obtained by examining only the feasible solutions to the subproblem. In the example, a total of 16 solutions would have to be investigated. Implementation-wise, efficient recursive procedures can be employed to generate the feasible solutions. Notice also the procedure lends itself to the application of heuristics. For example, just a few of the combinations could be generated and the best solution for these combinations determined.

Once one solution is obtained the new precedence relations established by this solution are added to the original network and the starting times are updated. The search continues for the next time period where a resource conflict occurs and a new subproblem is generated. The procedure repeats until all resource infeasibilities are resolved.

Although the procedure is still in its implementation phase and no significant tests have been performed, it is expected that the procedure will be able to find good solutions for problems of reasonable size. One problem having forty three activities has been solved by hand. The solution obtained by the decomposition procedure provided a makespan of 81 time units against an optimal solution of 74 time units. The procedure could also be further enhanced by allowing for user intervention in cases where critical activities must be scheduled with higher priorities. In this case, the user could specify the time at which such an activity must be scheduled.

9.0 Analysis Tasks

The following list of tasks summarizes the efforts and activities of the project:

1. Host and learn ESP.
2. Conduct literature review.
3. Begin work on heuristic-based optimization procedure.
4. Preliminary experimentation.
5. Fall briefing at MFSC: Application of cluster analysis and cluster scheduling.
6. Application of cluster analysis:
   a) Develop software
   b) Integration with ESP
   c) Identify factors
   d) Experiment with clusters and scheduling with ESP.

7. Develop software to implement the heuristic-based optimization scheduling procedure. (At this point, the algorithm addresses a subset of the experiment scheduling problem. This algorithm would have to replace some of ESP, or would have to be integrated into the logic of ESP.)
8. Analysis of experimental results.
9. Handling sequencing requirements
    a) Develop software to create depth-first spanning forest
    b) Develop interface with ESP database.
10. Handling concurrence
    a) Problems with selection groups (mandatory and necessary concurrence experiments must be in the same selection group).
11. Developed a three phase approach to the problem.
    a) Create three classes of experiments
    b) Develop software to assign experiments to the classes
    c) Develop a scheduling philosophy for each class.
12. Develop software to accomplish intra-class scheduling.
13. Experimentation with hybrid approach.
14. Summarize results for October briefing at MFSC.
15. Final documentation for Optimization algorithm .
16. Integrate project software for easy use with ESP.
17. Prepare Final Report.

10.0 Conclusions

The work performed in this project has accomplished two major tasks. First, a procedure was developed to prepare the selection lists that ESP needs so that it can produce an effective time line with a minimal amount of experimentation and manual effort. Schedules produced with our software produce schedules whose performance score is comparable with the time line created at MSFC, and our procedure requires no manual manipulation of the model file for the mission. The software has been integrated into a single menu-driven

package to facilitate selection group analysis, and the package operates directly upon the ESP data structures.

Second, we have developed a new optimization model for the resource constrained project scheduling problem, and this represents a direct contribution to the scheduling literature. While this model and its algorithm do not directly apply to the experiment scheduling problem, they do provide new insights into a special case of this problem and will be useful in their own right.

There is much more that could be done to further improve the scheduling performance of ESP, but most of this would involve integrating the selection list building with the inner workings of ESP. It is very difficult to estimate the degree of improvement that could be accomplished, especially in light of the anticipated significant increase in computational times and added complexity of the overall procedure. Perhaps the most significant gains in scheduling performance could be attained by standardizing the modeling process.

There is no question that the modeling process is somewhat of an art form; the requirements of experiments and their nature make it very difficult to create a single model template. That is, it seems impossible to convert the experiment specifications into the ESP model without a substantial amount of interpretation and trickery. Different modelers will surely produce different models, and the differences can be significant — especially in their impact on scheduling. For example, window length and location, targets, sequencing requirements, crew usage, resource usage and equipment usage are all very important in scheduling, but different modelers may make use of these in differing ways. Therefore, a standardized method to accomplish modeling that is designed to produce models that are compatible with the selection list creator and ESP is essential.

BIBLIOGRAPHY

[Bales] Balas, E., "An Additive Algorithm for Solving Linear Programs with Zero-One Variables," *Operations Research*, Vol. 13 (1965), 517-546.

[Balas] Balas, E., "Project Scheduling with Resource Constraints," *Applications of Mathematical Programming Techniques*, E. M. L. Beale, ed., American Elsevier (1970), 187-200.

[Bellman] Bellman, R., A. O. Esogbue and I. Nabeshima, *Mathematical Aspects of Scheduling and Applications*, Pergamon Press (1982).

[Carruthers] Carruthers, J. A. and Battersby, A., "Advances in Critical Path Methods," *Operational Research Quarterly*, Vol. 17, 4 (1966),359-380.

[Cooper] Cooper, D. F., "Heuristics for Scheduling Resource-Constrained Projects: An Experimental Investigation," *Management Science*, Vol. 22 (July 1976), 1186-1194.

[Davis] Davis, E. W. and G. E. Heidorn, "An Algorithm for Optimal Project Scheduling under Multiple Resource Constraints," *Management Science*, Vol. 17 (August 1971), B803-816.

[Davis] Davis, E. W. and J. H. Patterson, "A Comparison of Heuristic and Optimum Solutions in Resource-Constrained Project Scheduling," *Management Science*, Vol. 21 (April 1975), 944-955.

[Fendley] Fendley, L., "Toward the Development of a Complete Multi-Project Scheduling System," *Journal of Industrial Engineering*, Vol. 19 (October 1968), 505-515.

[Fisher] Fisher, M. L., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part II," in the *Proceedings of the Sympposium on the Theory of Scheduling and Its Applications*, North Carolina State University, May 15-17, (1972), 294-317.

[Fisher] Fisher, M. L., "Optimal Solution of Scheduling Problems Using Lagrange Multipliers: Part I," *Operations Research*, Vol. 21, (1973), 1114-1127.

[Gorenstein] Gorenstein, S., "An Algorithm for Project (Job) Sequencing with Resource Constraints," *Operations Research*, Vol. 20 (July/August 1972), 835- 850.

[Grone] Grone, R. D. and Mathis, F. H., "A Ranking Algorithm for Spacelab Crew and Experiments Scheduling," NASA/ASEE Summer Faculty Fellowship Program, (1980).

[Hastings] Hastings, N. A. J., "On Resource Allocation in Project Networks," *Operational Research Quarterly*, Vol. 23 (June 1952), 217-221.

[Held] Held, M. and Karp, R. M., "A Dynamic Programming Approach to Sequencing Problems," *Journal of the Society for Industrial and Applied Mathematics*, Vol. 10, 1 (1962), 196-210.

[Johnson] Johnson, T. J. R., An Algorithm for the Resource Constrained Project Scheduling Problem , Unpublished Ph.D. Thesis, Sloan School of Management, M.I.T., Aug. 1967.

[Kelley] Kelley, J. E., Jr., "The Critical-Path Method: Resources Planning and Scheduling," Chapter 21 in *Industrial Scheduling]* (John F. Muth and Gerald L. Thompson, eds.), (1963).

[Kurtulus] Kurtulus, I. and Davis, E. W., "Multi-Project Scheduling: Categorization of Heuristic Rules Performance," *Management Science*, Vol. 28, 2 (1982), 161-172.

[Lambourne] Lambourne, S., "Resource Allocation and Multi-Project Scheduling (RAMPS)-A New Tool in Planning and Control," *Computer Journal*, Vol. 5, 4 (1963), 300-304.

[Mathis] Mathis, F. H., "Mathematical Programming Techniques for Scheduling Spacelab Crew Activities and Experiment Operations," NASA/ASEE Summer Faculty Research Fellowship Program, (1981).

[Mize] Mize, J. H., A Heuristic Scheduling Model for Multi-Project Organizations , Unpublished Ph.D. Thesis, Purdue University, 1964.

[Pascoe] Pascoe, T. L., An Experimental Comparison of Heuristic Methods for Allocating Resources , Unpublished Ph.D. Thesis, Cambridge University, 1965.

[Patterson] Patterson, J. H., "Project Scheduling: The Effects of Problem Structure on Heuristic Performance," *Naval Research Logistics Quarterly*, Vol. 23 (March 1978), 95-123.

[Patterson] Patterson, J. H. and Huber, W. D., "A Horizon-Varying, Zero-One Approach to Project Scheduling," *Management Science*, Vol. 20, 6 (1974), 990- 998.

[Patterson] Patterson, J. H. and G. W. Roth, "Scheduling a Project under Multiple Resource Constraints: A Zero-One Programming Approach," *AIIE Transactions*, Vol. 8 (December 1976), 449-455.

[Patterson] Patterson, J. H., "A Comparison of Exact Approaches for Solving the Multiple Constrained Resource Project Scheduling Problem," *Management Science*, Vol. 30 (July 1984), 854-867.

[Petrovic] Petrovic, R., "Optimization of Resource Allocation in Project Planning," *Operations Research*, Vol. 16, 3 (1968),558-658.

[Phillips] Phillips, D. T. and Garcia-Diaz, A., *Fundamentals of Network Analysis*, Prentice-Hall, Inc. (1982).

[Plebani] Plebani, L. J., Jr., "A Heuristic for Multiple Resource Constrainted Scheduling," *Production and Inventory Management*, First Quarter (1981), 65-80.

[Pritsker] Pritsker, A. A. B., L. J. Watters and P. M. Wolfe, "Multiproject Scheduling with Limited Resources: A Zero-One Programming Approach," *Management Science*, Vol. 16 (September 1969), 93-108.

[Schrage] Schrage, L., "Solving Resource-Constrained Network Problems by Implicit Enumeration - Nonpreemptive Case," *Operations Research*, Vol. 18 (March / April 1970), 263-278.

[Slowinski] Slowinski, R., "Two Approaches to Problems of Resource Allocation among Project Activities - A Comparative Study," *Journal of the Operational Research Society*, Vol. 31 (August 1980), 711-723.

[Stinson] Stinson, J. P., E. W. Davis and B. M. Khumawala, "Multiple Resource Constrained Scheduling Using Branch and Bound," *AIIE Transactions*, Vol. 10 (September 1978), 252-259.

[Talbot] Talbot, F. B. and J. H. Patterson, "An Efficient Integer Programming Algorithm with Network Cuts for Solving Resource Constrained Scheduling Problems," *Management Science*, Vol. 24 (July 1978), 1163-1174.

[Talbot] Talbot, F. B., "Resource-Constrained Project Scheduling with Time- Resource Trade offs: The Nonpreemptive case," *Management Science*, Vol. 28 (October 1982), 1197-1210.

[Whitehouse] Whitehouse, G. E., and Tidwell, D., "Practical Computer Search Approaches to Project Management with Resource Constraints," *Proceedings of the 1980*

Spring Annual Conference of the American Institute of Industrial Engineers, Atlanta, Georgia, 335-339.

[Weiss] Weiss, H. J., "A Greedy Heuristic for Single Machine Sequencing with Precedence Constraints," *Management Science*, Vol. 27, 10 (1981), 1209- 1215.

[Weist] Weist, J. D., "A Heuristic Model for Scheduling Large Projects with Limited Resources," *Management Science*, Vol. 13, 6 (1967), B359-377.

APPENDIX 1

Literature Review

A wide variety of solution methods have been proposed for the solution of the resource constrained scheduling problem. Researchers have focused attention on heuristic methods which can easily generate feasible solutions to the problem. Kelley (1963) outlines two basic single pass (once an activity is scheduled, it is never rescheduled) heuristic methods: serial and parallel methods. The input is a list of activities ordered according to some priority scheme. In the serial method the list is scanned one activity at a time and the activity is scheduled in the first time slot for which all constraints are satisfied. The parallel method, a group of activities, determined by some criteria, is scheduled simultaneously. Several rules used to construct the priority list have been proposed. Most of them are based on information drawn from a critical path analysis of the network representation of the problem, and also on information concerning resources usage and availability. Grone and Mathis (1980) applied a serial method combined with an intricate multi-attribute priority scheme to the experiment schedule problem.

Whitehouse and Tidwell (1980) described an algorithm attributed to G.H.Brooks of Purdue University. This procedure, which became known as Brooks'algorithm, is similar in spirit to Kelley's (1963) parallel method. Activities are ranked according to the time each one controls through the network. At each time, t, set, called an allowable set, is defined containing activities that could potentially be scheduled based only on precedence constraint requirements. Activities from the allowable set are selected according to their ranks and as many activities as possible are scheduled. Time is then advanced to the next completion time and the procedure is repeated until all activities are scheduled. In addition, Whitehouse and Tidwell (1980) proposed changes in the ranking procedure in order to take into account resource usage. Plebani (1981) suggests a generalization for the case of multiple resources.

Several studies have been devoted to testing the effectiveness of priority rules. Cooper (1976), Fendley (1966), Mize (1964), Pascoe (1965) and Patterson (1976) compared the performance of several priority rules. Davis and Patterson (1975) tested the effectiveness of priority rules for a set of test problems when compared to optimal solutions. Unfortunately, the results reported in the literature are not conclusive; sometimes they are even contradictory. On one point, however, most researchers have agreed; the performance of priority rules is very much problem dependent. Kurtulus and Davis (1982) and Patterson (1976) developed regression type procedures which take into account the differences in problem characteristics when assessing performance of heuristics.

Several computerized versions of heuristic methods have become commercially available. Early in the 60's, Lambourne (1963) reported on the development of RAMPS. Wiest (1967) developed SPAR-1 and Phillips and Garcia-Dias (1981) describe several other systems.

While heuristic procedures are attractive from a practical standpoint, there is usually no way to guarantee the quality of the solution. This fact has led many researchers to the search for optimization procedures. Several such procedures have appeared in the

literature. Perhaps the first integer linear programming formulation of the resource constrained scheduling problem was given by Wiest (1963). Pritsker, Watters and Wolfe (1969) presented a new integer programming formulation which accommodates a wide range of real-world situations including due dates, activity preemption, resource substitutability and concurrency. The major drawback of this kind of approach is the necessity to define a 0-1 variable for each possible combination of activity and completion time, implying that even for small problems, a large number of variables must be defined. Each variable assume a value 1 in the period its corresponding activity is started. Mathis(1981) presented an adaptation of the previous model applied to the NASA experiment scheduling problem concluding that millions of variables would have to be defined for such a problem.

Patterson and Huber (1974), using basically the same formulation proposed by Pritsker et al. (1969), present three procedures: minimum bounding algorithm, maximum bounding algorithm, and a search algorithm which combines the two previous procedures. In the minimum bounding algorithm, an initial lower bound on completion time is obtained. This value is decreased by one at each iteration, and a new 0-1 programming problem is solved. If the current solution is feasible, it is also optimal. Otherwise the procedure continues until feasibility is attained. Results based on a set of test problems showed that the minimum bounding algorithm was the most effective among the three.

Again, considering a formulation similar to the one by Pritsker et al. (1969), Patterson and Roth (1976) developed an implicit enumeration procedure based on Balas'(1965) additive algorithm. The procedure seeks minimization of the completion time of the last activity scheduled. Variables are ordered according to activity label and time period, and considered for augmentation according to that order. Only one of the variables associated to activity i can assume the value 1 at a given time. Precedence constraints can easily be handled by this scheme. Whenever a complete feasible solution which reduces the project completion time by t time units is found, the t right-most variables corresponding to each activity can be eliminated. A complete feasible solution is achieved whenever one of the variables corresponding to activity n is set equal to 1. Optimality is attained whenever all possible combinations are implicitly verified.

In a more recent version of this algorithm, Talbot and Patterson (1978) use the concept of network cuts in a procedure to improve the fathoming process of partial solutions. Talbot (1982) further modified the procedure by considering alternative modes of performing an activity.

An attempt to employ pure linear programming to the resource constrained scheduling problem was reported by Weglarz, Blazewicz, Cellary and Slowinski (1977). The formulation, presented in detail in another paper by Slowinski (1980), has a serious shortcoming since a pre-determined ordering for the start of each activity is assumed. In addition, preemption is allowed in this formulation. In a subsequent paper, Slowinski (1981) generalized the procedure for multi-objective criteria.

Other types of implicit enumeration procedures which avoid the use of binary variables also exist. Johnson (1967) developed a branch and bound procedure for the one resource problem which was later enhanced by Patton (1968) and, more recently, extended to the multiple resource case by Stinson, Davis and Khumawala (1978). In these procedures, the nodes in the branch and bound tree correspond to feasible partial schedules. Lower bounds

on project completion time can be determined based either on precedence constraints, resource constraints, or a combination of both. Rules are developed which can be used for dominance pruning. Other papers by Hastings (1975) and Schrage (1970) present methods which are similar in spirit to Johnson's (1967).

In a procedure due to Davis and Heidorn (1971), the resource constrained scheduling problem is reduced to finding the shortest path in a network which the authors call A-network. In order to construct such a network, all feasible subsets (set of activities such that if activity i is in this set, all its predecessors also are) are systematically generated. The nodes of the A-network correspond to feasible subsets.

Patterson (1984) performed a comparative study of the three methods developed respectively by Davis and Heidorn (1971), Stinson, Davis and Khumawala (1978) and Talbot and Patterson (1978). He concludes that each procedure was superior to the others on a specific class of problems.

The first optimization procedure which attempts to exploit problem structure was proposed by Fisher (1973). He assumed that a set of independent activities must be performed, each activity being represented by a series of precedence related steps. Fisher used Lagrange multipliers to dualize resource constraints and recognized that the resulting Lagrangean problem can be decomposed into independent problems for each activity, since precedence constraints do not interact among activities. In another paper, Fisher (1972) uses the solution to the Lagrangean problems as lower bounds in a branch and bound procedure to obtain the optimal solution. In this procedure, nodes correspond to derived Lagrangean problems and branching corresponds to updating the values of the Lagrange multipliers. Unfortunately, the method only works for objective functions which can be written as nondecreasing functions of the activity completion times. This restriction eliminates the problem of minimizing the completion time of the last activity scheduled.

A very elegant formulation of the resource constrained scheduling problem was proposed by Balas (1970). The idea is to take the network representation of the precedence constraints and add pairwise disjunctive arcs (two arcs in reverse directions, at most one being active at a time) between any two activities that share the same resource. Each pairwise disjunctive arc may be in one of the following states: forward arc is active, backward arc is active, or both arcs are inactive. A selection is a set which indicates the state of all disjunctive arcs. The problem reduces to finding the feasible selection which minimizes the critical path in the network. Balas (1970) developed the concept of internal stability which is used for testing feasibility. Gorenstein (1972) further developed the ideas of Balas (1970) and showed that a maximun flow algorithm can be used as feasibility test. The most attractive feature of Balas' formulation is that model size is independent of the number of time periods in the planning horizon.

Perhaps the least successful optimization attempt to solve this problem has been dynamic programming. Some researchers (Held and Karp 1963 and Petrovic 1968) have formulated different versions of scheduling problems, none of them applicable to the resource constrained scheduling problem.

APPENDIX 2

GLOSSARY OF TERMS

The terminology used here is adapted from that used within ESP, rather than that used in the broader context of the experiment scheduling problem.

Experiment : A functional objective to be performed; is the same thing as a model.

Performance : An instance of an experiment. Each experiment has a requested number of performances or instances.

Distinguished Steps : If an experiment has a start-up or shut- down step, these will be called distinguished steps because these do not have to be accomplished by all performances.

Sequencing Requirement : An experiment has a sequencing requirement if it has a predecessor experiment; that is, the experiment cannot be scheduled unless its predecessor has already been scheduled.

APPENDIX 3

SUMMARY OF EXPERIMENTAL ANALYSIS

ESP Scheduling Data

Timeline File Name: TLA.DAT

Schedule Grade: 75.4

Selection Order Description: The first schedule produced came strictly from the sequencing procedure. The program SEQ.FOR builds precedence trees based on the sequencing requirements set forth in the model file. The selection list is separated into three groups. The first hopper contains those experiments with other experiments dependent upon them being scheduled before they can be scheduled. The second hopper contains all experiments with mandatory or necessary concurrence. The third, and last, hopper contains all remaining experiments not in the previous hoppers.

All of the hoppers were selected by the fixed selection method.

ESP Scheduling Data

Timeline File Name: TLB, TLC, TLD.DAT

Schedule Grade: 75.4, 75.7, 74.9

Selection Order Description: These three schedules all use the random selection method for the first two hoppers to schedule. The third hopper was selected by the fixed method. Three different random number seeds were used, all producing similar results. They were 29387, 81585, and 00001, respectively.

The purpose of this test was to determine the effects, if any, that the random number seed might have on the schedule. It seems that different random number seeds do not effect the result significantly.

ESP Scheduling Data

Timeline File Name: TLE.DAT

Schedule Grade: 77.6

Selection Order Description: Beginning with this schedule, The third hopper has been processed further. The first two hoppers remain the same (ie., those experiments with other experiments dependent upon them in hopper 1 and those with mandatory concurrence requirements in hopper 2).

The last hopper has been run through the clustering software developed previously. The experiments have been separated according to the number of performances required. Three groups have been produced: experiments with less than five performances required, experiments with five to ten required, and experiments with greater than ten required. This results in a total of five hoppers selected from for the schedule. The intent here was to see what effects grouping technology might have on the schedule. The only difference between this schedule and the schedule for TLA.DAT is the manipulation of hopper three.

ESP Scheduling Data

Timeline File Name: TLF.DAT

Schedule Grade: 71.8

Selection Order Description: This schedule reverses the previous schedule run (TLE.DAT). Those experiments with a large number of performances required run first (hopper 3), descending down to those experiments with less than five performances required run last (hopper 5). This showed how scheduling using the number of performances required by an experiment as the main criterion for selection of the experiments to schedule first.

ESP Scheduling Data

Timeline File Name: TLG.DAT

Schedule Grade: 76.1

Selection Order Description: The third hopper from the set of three was again run through the clustering software, but with a different grouping criterion. In order to test the importance of crew selection requirements, the clustering was performed on this resource. Crew selection was divided into three categories. Strict selection requirements, not strict selection requirements, and no crew requirements. Strict selection requirements is defined as those experiments that require specific crew members as resources. If some flexibility is possible (eg., of two crew members listed, pick one), the experiment was labelled

as having non-strict selection requirements. The last category was all those experiments that required no crew members at all. ESP was run with the experiments with strict selection requirements selected first, followed by the experiments with non-strict selection requirements, and, lastly, the experiments with no crew requirements at all.

ESP Scheduling Data

Timeline File Name: TLH.DAT

Schedule Grade: 73.6

Selection Order Description: This schedule reverses the previous schedule produced (TLO.DAT) by attempting to schedule those experiments with less strict selection requirements first, followed by more and more strict selection requirements. There were a total of five groups for crew selection, for a total of seven hoppers used for this resource test.

ESP Scheduling Data

Timeline File Name: TLI.DAT

Schedule Grade: 75.0

Selection Order Description: This and the next schedule separated resource requirements according to required targets for the experiments. Again, the first two hoppers contain the experiments with other experiment dependencies and with mandatory concurrence, respectively. The targets were grouped according to six categories; the first five according to specific targets and the last with no target requirements.

A list of targets considered critical was obtained from NASA. Five levels of criticality were formed from this list that could be used for clustering purposes. This schedule selected most critical targets first, less critical targets next, and so on, until all levels of target criticality were scheduled.

ESP Scheduling Data

Timeline File Name: TLJ.DAT

Schedule Grade: 65.0

Selection Order Description: This schedule selected the experiments with less critical targets first, ascending to experiments with greater target criticality.

The great difference in scores between this schedule and the previous schedule indicates some degree of importance in the selection order of experiments with target requirements.

ESP Scheduling Data

Timeline File Name: TLK.DAT

Schedule Grade: 71.2

Selection Order Description: This schedule and following schedules return to dividing the third hopper into three hoppers according to the number of performances required. Hopper three is now the experiments that require less than five performances, hopper four is those that require five to ten, and hopper six is those that require more than ten performances. The second hopper has also been refined further. It consists of those experiments with mandatory concurrence. In this schedule, hopper two was also sorted by number of required performances in ascending order. However, because of the concurrence requirements, they must remain within one hopper.

ESP Scheduling Data

Timeline File Name: TLL.DAT

Schedule Grade: 77.6

Selection Order Description: This schedule is the same as the previous schedule (TLM.DAT), with one exception. The random number used for the previous schedule failed to schedule one of the experiments in hopper one; these are the experiments that must be scheduled before others may succeed them. Thus, a different random number seed was used for this run.

ESP Scheduling Data

Timeline File Name: TLM.DAT

Schedule Grade: 74.3

Selection Order Description: This schedule goes back to the original sequenced order used in the first schedule (TLA.DAT). Without changing the third hopper as in previous schedules, the second hopper is refined. Those experiments with mandatory concurrence (ie, those in the second hopper) have been ordered according to the width of the time windows. Those that have a very short time window to schedule are attempted first, followed by those that do not have as strict window requirements.

ESP Scheduling Data

Timeline File Name: TLN.DAT

Schedule Grade: 69.2

Selection Order Description: This schedule has taken the previous scheduling technique and expanded upon it. The second hopper has been divided into two hoppers. The first attempt at scheduling the mandatory concurrence experiments requests only half of the performances required by each experiment in the second hopper. The second half of the required performances is requested in hopper four (after the remaining experiments have been scheduled).

ESP Scheduling Data

Timeline File Name: TLP.DAT

Schedule Grade: 76.0

Selection Order Description: This and the next timeline (TLQ) go back to grouping the third hopper as in timelines TLE through TLJ. The new clustering criteria is according to the number and gap of the time windows in which an experiment can be scheduled. This schedule ordered the windows by smallest window to largest and multiple windows last.

ESP Scheduling Data

Timeline File Name: TLQ.DAT

Schedule Grade: 74.8

Selection Order Description: This schedule changes the order of the windows from those experiments with multiple windows first, followed by those with large windows descending to those with the smallest windows last.

ESP Scheduling Data

Timeline File Name: TLR.DAT

Schedule Grade: 77.4

Selection Order Description: This and the next timeline order the third hopper according to three properties used for clustering. The three properties are the number of performances, the size of scheduling windows, and the targets used. The clusters were grouped into the following hoppers: 1) 1, 2) 7,9, and 11, 3) 3,4,5, and 6, and 4) 2,8, and 10.

ESP Scheduling Data

Timeline File Name: TLS.DAT

Schedule Grade: 72.5

Selection Order Description: This schedule reverses the clustering order used in the previous schedule. The last five hoppers are simply in the reverse order of that used in TLR.DAT.

APPENDIX 4

CLUSTER ANALYSIS

The first step in the cluster analysis is to compute the similarity matrix and to form component packs. To compute the similarity matrix the factor vector of each pair of experiments (i,j) are compared. If i and j use the same factors then these two experiments are packed together. All experiments within a pack are considered as one experiment for the remainder of the analysis. While the packing is performed, the similarity index SIM(i,j) for each pair of experiments (i,j) is computed according to the rule:

$$SIM(i,j) = \frac{no.\ factors\ in\ common}{Total\ factors\ used\ by\ i\ and\ j}$$

The next step in the process is to sort the packs in preparation for clustering. Based on the requested value of k, the packs are sorted into descending order by their k-th largest similarity indices. The sensitivity parameter k influences cluster formation in only an indirect manner. k = 1 produces the most clusters because it is more difficult for packs to be combined together to form clusters. As k increases, fewer clusters are formed at the expense of forming more clusters. The analyst should experiment with different k values to gain some insight into the properties and characteristics of the specific case study.

The actual cluster formation begins at this point. Packs are analyzed one at a time in the order in which they have been sorted. Each pack is either assigned to a current cluster (if this pack is relatively similar to those packs already in the cluster) or creates a new cluster (if no relatively similar clusters exist). If a pack is sufficiently similar to two or more existing clusters, these clusters may be combined into one large cluster. The following steps specify the details of the clustering algorithm.

Step 1:

The highest ordered pack (KMIN), based upon its kth largest similarity index (PMIN) is the next (or first) pack to be assigned to a cluster.

Step 2a:

If the packs similarity (PMIN) is greater than or equal to the largest inter-cluster similarity between all existing clusters (DMIN = the maximum element in GRPSIM, the array of inter-group similarities), then the pack will either start a new pack or be assigned to an existing pack. Logically, KMIN is more similar to at least k other packs than are any existing clusters to each other. Therefore, it should go toward packs similar to itself.

Step 3a:

Compare pack KMIN to all assigned packs and find the pack and its cluster with the greatest similarity. This similarity must be larger than PMIN. If a pack (and cluster) is found, then assign pack KMIN to that cluster. If a pack is not found, pack KMIN starts a new cluster. Logically, there are k packs to which pack KMIN has a similarity of at least PMIN. If one of these packs is found in a cluster, then pack KMIN can be assigned there. The effect of the value of k is that a small value makes it more difficult to find a similar pack; therefore encouraging the formation of new clusters.

Step 4a:

Since a new pack has been added to a cluster, the intra-group similarity and inter-group similarities must be recalculated. The average intra-group similarity, TOT, for each cluster is calculated as follows:

$$TOT(g) = \sum_{i \in g} \frac{SIM(KMIN, i)}{number\ of\ packs\ in\ group\ g}$$

The inter-cluster similarity, GRPSIM(ialc,i) between groups i and ialc, where ialc is the cluster number that KMIN was assigned to.

$$GRPSIM(ialc, i) = \frac{GRPSIM(ialc, i) \times (no.\ of\ packs\ in\ ialc)] + TOT(i)}{no.\ packs\ in\ ialc + 2}$$

Step 5a:

GO TO STEP 8.

Step 2b:

If PMIN is less than the greatest inter-cluster similarity, DMIN, then the clusters creating that value, DMIN, will be combined to form a single cluster. Logically, these clusters are more similar to each other than the pack KMIN is to all but k other packs. Here, a large k will tend to drive PMIN to a smaller value, and thus encouraging the aggregation of clusters.

Step 4b:

Go to subroutine MCPAK to combine factors in the newly formed cluster.

Step 5b:

Transfer the packs of the merging cluster into the receiving cluster.

Step 6b:

Since the clusters were fused, the inter-cluster similarities (in GRPSIM) must be recalculated. The new cluster's inter-similarity is calculated by condisering the previous inter-similarity of each of the combined clusters with all other clusters and the number of packs in each of the joined clusters. The calculation is given by:

$$GRPSIM(a,b) = \frac{[GRPSIM(a1,b) \times (\#of\ packs\ in\ a1)] + [GRPSIM(a2,b) \times (\#packs\ in\ a2]}{\#packs\ in\ a1 + \#packs\ in\ a2}$$

where a and b are cluster numbers, and a1 and a2 combined to form a.

Step 7b:

Go to step 8.

Step 8:

Find the two clusters with the greatest inter-cluster similarity (GRPSIM). Set this similarity value to DMIN, which will be used to identify the need to combine clusters in future iterations.

APPENDIX 5

USER'S GUIDE

The selection list sequencing software is a set of programs that may be used to modify the selection order of experiments in an ESP model file. The software offers a flexible and simple way to organize the experiments in such a way as to produce an experiment schedule that grades very favorably when compared to previous methods of repeated trial runs. After only one execution of the selection list sequencing software, a fairly "good" schedule will result that may then be further optimized with the ESP selection list editor. This greatly reduces the time required to produce a usable experiment schedule.

The entire selection list sequencing software is written in VAX-11 Fortran version 3.0. With the exception of the menu program, the software programs are coded as subroutines and are part of an object library used for linking purposes. The main (and menu) program is the file ATM.FOR and the object library is CLSTR.OLB. The sequencing software is integrated with parts of the ESS support package supplied by MSFC. Thus, the executable code must also be linked with this object library.

The sequencing software is delineated into definite groups according to function. None of the functions are required—all may be omitted if desired. However, those functions that are executed must be processed in a required sequence.

There are six different options that exist in the selection list sequencing software. Each will be discussed in turn, followed by remarks concerning the execution order requirements. A description of the purpose of each function, how the code processes the data, and how to execute the code is given. In conclusion, a brief discussion of problems with the software that may arise in the future is included.

To execute the selection list sequencing software, simply run the program ATM.EXE from the VAX/VMS command language prompt. A menu of the six available functions and an exit option will be displayed on the terminal. The six options are [1] Create Sequenced Hoppers, [2] Sequence Hopper 1, [3] Cluster Hopper 3, [4] Print Cluster Report, [5] Group Clusters into Hoppers, and [6] Write Selection List to Model File.

[1] Create Sequenced Hoppers

If any of the selection list sequencing software functions are to be used, the sequencing function (option 1) must be executed prior to any others. The models of the experiments are constrained by default specifications in the model file. Certain models are required to be scheduled before other, dependent models may be scheduled. This function accesses the experiment model file and orders the models according to three categories. The first category is those models that must precede one or more other models in the schedule. The second category is those models with mandatory or necessary concurrences. This is done in order to keep the concurrent models within the same hopper for scheduling. The last category is any models not included in the first two categories. Note that any models with precedence requirements as well as concurrence requirements will force the required concurrent model into the first category.

The subprogram to carry out this funciton is SEQ.FOR. This program traverses a depth-first spanning tree representing the sequencing structure of the entire model file. At each node, a tally is kept of all models that will succeed that node. After one pass through the file, all models with successors will have a non-zero tally. Another pass is then performed that will decrement the tally of those nodes that have no successors and no concurrence requirements.

At this point, the three categories have been defined. They are: (1) nodes, or models, with positive tallies and having successors, (2) nodes with a tally equal to zero and having mandatory or necessary concurrence constraints, and (3) nodes having negative tallies with neither of the previous requirements. An array containing this information is then sorted according to tally and, within tally, by experiment number.

A final pass is performed to write a file for each category. The files correspond to the categories defined above and are named HOP1.DAT, HOP2.DAT, and HOP3.DAT. As the names imply, the files contain the experiment numbers of those models to be written into the first three hoppers of the selection list.

These simple steps complete the "Create Sequenced Hoppers" function of the selection list sequencing software. Once this function has been performed, the other optional functions may be performed. The three files created here may be further processed at the user's option upon returning to the menu.

[2] Sequence Hopper 1

The second function, "Sequence Hopper 1", is an optional function that further sequences the experiments in hopper 1. These are the models that must be scheduled prior to other models and are contained in the file HOP1.DAT.

The subprogram creates a matrix of $n \times n$ dimension, where n is the number of models read in from the file. Each cell in the matrix is a Boolean value (0 or 1) that sets the sequence dependency of the models in that row and column. A value of 1 in row i, column j indicates the model in row i has precedence over the model in column j.

A precendence will be established for all models in the matrix. First, any default dependencies (the same used in option 1) are marked, including transitive precendence. For example, if y follows x and z follows y, then z must also follow x. The cells in row x, columns y and z will be marked with 1.

If no default precedence exists, a precedence is determined from a cost function. A cost is calculated for each model using some calculation. The calculation used in this version of the software is simply the width of the smallest window available to the model for scheduling. Other resource comparison calculations may be used by making minor changes to the program.

After all sequencing dependencies have been established, the number of 1's in each column is summed. The column tally with the lowest value is the new model to be scheduled first. The next lowest value is the second model to be scheduled, etc. The subprogram writes the new list of models back into HOP1.DAT to complete the sequencing for hopper 1.

[3] Cluster Hopper 3

The third option of the selection list sequencing software, "Cluster Hopper 3," groups similar models into subgroups. The models listed in HOP3.DAT are grouped together according to a similarity index value that is entered by the engineer. Options [4] and [5] are subsequent steps of the clustering option and will be discussed later. The result of these three steps will be the partitioning of HOP3.DAT into one or more hopper lists to be

written to the selection list. That is, this hopper will be subdivided into several hoppers for ESP scheduling.

First, a file is built containing the model (or experiment) numbers from HOP3.DAT and their respective "machines" used. The "machines" are names of resource categories defined in the subprogram ROUTMOD.FOR. For example, if a model required five or less performances to be scheduled, it was assigned the "machine" Perf1. If a model required six to ten performances to be scheduled, it was assigned the "machine" Perf2. For eleven or more performances, a "machine" of Perf3 was assigned. Other machines, or resources, used for categorization were targets and window sizes required by a model.

Using this machine file as input, a matrix of all models and the machines used by that model is created. Another matrix is created that keeps track of which models require identical sets of machines. These exact models are combined into what are known as packs. An output listing of this file may be obtained by printing the file PAK.RPT from VMS.

At this point, the engineer is prompted to input a "k" value. This is the similarity index. The similarity index is used to combine the packs according to the degree of similarity of the packs. For example, if a similarity index of two is used, all packs with at least two identical machines will be combined into a cluster. Generally, the higher the k value used, the fewer clusters will be formed. A cluster report file, CLU.RPT, is produced that should be printed (option 4) and examined. Similar clusters will be further combined into hoppers before being written to the model file selection list.

[4] Print Cluster Report

The cluster report file should be printed by the option "Print Cluster Report" immediatly following option 3. This is a simple Fortran program that conveniently prints the cluster report without leaving the selection list sequencing software menu. The file will be printed to "LPA0:". If some other output device is desired, this logical name may be reassigned in VMS.

This report should be examined by the engineer to determine which clusters should be combined into hoppers. For example, all clusters using Perf1 (for less than five performances)

might be grouped together. Once the grouping requirements have been established, the clusters can be combined into hopper lists with option 5.

[5] Group Clusters into Hoppers

The third, and last, step of the clustering of hopper three is "Group Clusters into Hoppers." This step allows the engineer to interactively assign the clusters to hoppers. To properly complete this step, the engineer should have clustered the experiments in HOP3.DAT (option 3) and printed the cluster report (option 4).

This subprogram will display upon the terminal a template showing the numbered clusters and the current hopper to which the specified cluster will be assigned. The cluster numbers will be displayed horizontally across the screen up to a maximum of twenty. The engineer is expected to enter the number of a cluster to be placed into the hopper. As a cluster number in entered, the models in that cluster will be written into a list. If a zero is entered, a flag will be written to the list signifying the end of that hopper. The hopper count is incremented so that new cluster numbers may be specified to be placed into the new hopper. This process continues until all clusters have been assigned.

At this time, the list containing the model assignments to the different hoppers is written back to HOP3.DAT. Although the filename still implies hopper three, the data is separated into hoppers by flags in the file. This effectively has broken the single hopper into several hoppers according to the resource requirements specified in the clustering process. The clustering of hopper three is now complete.

[6] Write Selection List to Model File

This option is used to write the new selection list to the model file. The subprogram was supplied by NASA and modified so that it could be integrated with the selection list sequencing software.

The models are written to the selection list of the model file in the order HOP1.DAT, HOP2.DAT, and HOP3.DAT. The first two files correspond to single hoppers while the third may correspond to one or more hoppers. The subprogram reads the first two files

and writes them to the selection list. When the third file is read, a new hopper is created each time the end-of-hopper flag is encountered. If no such flags exist (eg., HOP3.DAT may not have been clustered), all models in the file will be included in the same hopper.

Execution Order

The execution order of the selection list sequencing software options is important. Option 1, "Create Sequenced Hoppers," must always be executed first. Option 6, "Write Selection List to Model File," must always be executed last. If option 6 is not executed at all, no updating of the selection list of the model file will occur at all. Thus, if the engineer is not satisfied with some intermediate result, updating the model file may be omitted entirely.

Options 2 through 5 are used to further refine the sequencing in hoppers one and three. Option 2 sequences hopper one and may be executed anytime after option 1 but before option 6. Options 3 through 5 sequence hopper three and, if used, must be executed in ascending order. Again, this set of options may be executed anytime after option 1 but before option 6.

Future Problems

Specific problems that may arise with the software are difficult to anticipate; however, it may be of some use to speculate. In most cases, no data of great importance will be destroyed if a program aborts during execution. The user may simply restart the program and try again. No extensive error checking is done in this software, so invalid data may cause premature program termination. A value entered by the user is an example. The software might exceed the bounds of some work array dimension. This software was tested on only one data set. It is hoped that ample space has been reserved, but a large number of experiments may cause problems.

If these or other problems do arise, the source code has been well-documented. In some cases, each line of code in an entire program has its own comment. This will aid in the modification of any of the source code should it become necessary.

APPENDIX 6

SAMPLE DIALOGUE

```
$ RUN ATM


*********************************************
*     SELECTION LIST SEQUENCING SOFTWARE    *
*           Texas A&M University            *
*********************************************


      [1] Create Sequenced Hoppers
      [2] Sequence Hopper 1
      [3] Cluster Hopper 3
      [4] Print Cluster Report
      [5] Group Clusters into Hoppers
      [6] Write Selection List to Model File
      [E] Exit from program


               Enter Choice
1

DEFAULT MODEL FILE IS     SPL.MOD
ENTER NEW FILE NAME>
SUCCESSFULLY OPENED DLA0:[ESP.ESP]SPL.MOD;1




*********************************************
*     SELECTION LIST SEQUENCING SOFTWARE    *
*           Texas A&M University            *
*********************************************


      [1] Create Sequenced Hoppers
      [2] Sequence Hopper 1
      [3] Cluster Hopper 3
      [4] Print Cluster Report
      [5] Group Clusters into Hoppers
      [6] Write Selection List to Model File
      [E] Exit from program


               Enter Choice
2

DEFAULT MODEL FILE IS     SPL.MOD
ENTER NEW FILE NAME>
SUCCESSFULLY OPENED DLA0:[ESP.ESP]SPL.MOD;1
```

```
*********************************************
*     SELECTION LIST SEQUENCING SOFTWARE    *
*           Texas A&M University            *
*********************************************

      [1] Create Sequenced Hoppers
      [2] Sequence Hopper 1
      [3] Cluster Hopper 3
      [4] Print Cluster Report
      [5] Group Clusters into Hoppers
      [6] Write Selection List to Model File
      [E] Exit from program

                Enter Choice
3

DEFAULT MODEL FILE IS     SPL.MOD
ENTER NEW FILE NAME>
SUCCESSFULLY OPENED DLA0:[ESP.ESP]SPL.MOD;1

                          Calculating Similarity Indices

                              Forming Component Packs



                     Writing the pack report PAK.RPT to disk.


          Input value of k (0 < k <=  32 ) :

2


                     Sorting to Find Kth Largest Similarities


                                Forming the Clusters
```

```
*********************************************
*    SELECTION LIST SEQUENCING SOFTWARE     *
*          Texas A&M University             *
*********************************************


     [1] Create Sequenced Hoppers
     [2] Sequence Hopper 1
     [3] Cluster Hopper 3
     [4] Print Cluster Report
     [5] Group Clusters into Hoppers
     [6] Write Selection List to Model File
     [E] Exit from program


              Enter Choice
4




*********************************************
*    SELECTION LIST SEQUENCING SOFTWARE     *
*          Texas A&M University             *
*********************************************


     [1] Create Sequenced Hoppers
     [2] Sequence Hopper 1
     [3] Cluster Hopper 3
     [4] Print Cluster Report
     [5] Group Clusters into Hoppers
     [6] Write Selection List to Model File
     [E] Exit from program


              Enter Choice
5
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=================================================
   Hopper used

   Enter cluster number to include in Hopper  1
    or 0 (zero) to end Hopper
1



Group Number      1  2  3  4  5  6  7  8  9 10 11
=================================================
   Hopper used    1

   Enter cluster number to include in Hopper  1
    or 0 (zero) to end Hopper
7



Group Number      1  2  3  4  5  6  7  8  9 10 11
=================================================
   Hopper used    1                 1

   Enter cluster number to include in Hopper  1
    or 0 (zero) to end Hopper
9



Group Number      1  2  3  4  5  6  7  8  9 10 11
=================================================
   Hopper used    1                 1     1

   Enter cluster number to include in Hopper  1
    or 0 (zero) to end Hopper
0
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1                 1     1

   Enter cluster number to include in Hopper  2
    or 0 (zero) to end Hopper
11
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1                 1     1     2

   Enter cluster number to include in Hopper  2
    or 0 (zero) to end Hopper
0
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1                 1     1     2

   Enter cluster number to include in Hopper  3
    or 0 (zero) to end Hopper
6
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1              3  1     1     2

   Enter cluster number to include in Hopper  3
    or 0 (zero) to end Hopper
0
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1                 3  1     1     2


   Enter cluster number to include in Hopper  4
    or 0 (zero) to end Hopper
2



Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4              3  1     1     2


   Enter cluster number to include in Hopper  4
    or 0 (zero) to end Hopper
3



Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4  4           3  1     1     2


   Enter cluster number to include in Hopper  4
    or 0 (zero) to end Hopper
5



Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4  4     4  3  1     1     2


   Enter cluster number to include in Hopper  4
    or 0 (zero) to end Hopper
0
```

```
Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4  4     4  3  1     1     2


   Enter cluster number to include in Hopper  5
    or 0 (zero) to end Hopper
4



Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4  4  5  4  3  1     1     2


   Enter cluster number to include in Hopper  5
    or 0 (zero) to end Hopper
8



Group Number      1  2  3  4  5  6  7  8  9 10 11
=============================================
   Hopper used    1  4  4  5  4  3  1  5  1     2


   Enter cluster number to include in Hopper  5
    or 0 (zero) to end Hopper
10



  Formation of cluster list complete...
```

```
*********************************************
*     SELECTION LIST SEQUENCING SOFTWARE    *
*           Texas A&M University            *
*********************************************


      [1] Create Sequenced Hoppers
      [2] Sequence Hopper 1
      [3] Cluster Hopper 3
      [4] Print Cluster Report
      [5] Group Clusters into Hoppers
      [6] Write Selection List to Model File
      [E] Exit from program


                Enter Choice
6

DEFAULT MODEL FILE IS     SPL.MOD
ENTER NEW FILE NAME>
SUCCESSFULLY OPENED DLA0:[ESP.ESP]SPL.MOD;1



*********************************************
*     SELECTION LIST SEQUENCING SOFTWARE    *
*           Texas A&M University            *
*********************************************


      [1] Create Sequenced Hoppers
      [2] Sequence Hopper 1
      [3] Cluster Hopper 3
      [4] Print Cluster Report
      [5] Group Clusters into Hoppers
      [6] Write Selection List to Model File
      [E] Exit from program


                Enter Choice
E
```

APPENDIX 7

SAMPLE CLUSTER OUTPUT

COMPONENTS PACKS

PACK NO	COMPONENTS IN PACK		M/CS NEEDED
1	000000001		Perf2 WSmall
2	000000002	000000004	
	000000005	000000006	
	000000007	000000091	Perf3 WSmall
3	000000003	000000034	
	000000037		Perf3 TgtIV WSmall
4	000000008	000000009	
	000000010	000000011	
	000000012	000000013	
	000000014	000000015	
	000000016	000000017	
	000000018	000000019	
	000000020	000000021	
	000000022	000000068	
	000000069	000000070	
	000000082	000000085	
	000000086	000000087	
	000000089	000000144	
	000000181		Perf1 WSmall

5	000000023	000000024			
	000000060	000000098			
	000000099	000000132			
	000000151	000000165			
	000000182	000000183			
			Perf3	WHuge	
6	000000025	000000062			
	000000064	000000088			
	000000145	000000157			
			Perf1	WHuge	
7	000000030	000000031			
	000000113	000000123			
	000000126	000000127			
			Perf1	TgtIV	WHuge
8	000000032				
			Perf2	TgtIV	WMult
9	000000033	000000035			
	000000052	000000053			
	000000065	000000081			
	000000084	000000131			
			Perf1	TgtIV	WSmall
10	000000036	000000038			
	000000039	000000040			
	000000041	000000042			
	000000043	000000171			
			Perf2	TgtIV	WSmall

20	000000090				
			Perf3	WMult	
21	000000107	000000109			
	000000118	000000124			
	000000129				
			Perf2	TgtIV	WHuge
22	000000108	000000114			
	000000115				
			Perf3	TgtII	WHuge
23	000000110	000000111			
	000000116	000000117			
	000000119	000000125			
			Perf1	TgtII	WHuge
24	000000112	000000170			
	000000176	000000177			
	000000178				
			Perf2	TgtI	WHuge
25	000000120	000000122			
			Perf2	TgtII	WHuge
26	000000148	000000152			
	000000153	000000154			
	000000179	000000180			
			Perf2	WHuge	

11	000000044 000000147	000000045	Perf1	TgtIV	WLarge
12	000000054 000000074 000000079	000000071 000000075	Perf1	TgtIII	WSmall
13	000000058		Perf3	TgtIV	WMult
14	000000061	000000063	Perf3	TgtIV	WHuge
15	000000066 000000080	000000073	Perf1	WMult	
16	000000067	000000072	Perf1	TgtIV	WMult
17	000000076		Perf1	TgtIV	WMedum
18	000000077	000000083	Perf1	TgtI	WSmall
19	000000078		Perf1	TgtIII	WMult

27	000000149		Perf1	TgtIII	WLarge
28	000000150	000000155	Perf1	TgtV	WHuge
29	000000156		Perf2	TgtIII	WHuge
30	000000161		Perf1	WLarge	
31	000000162		Perf2	WMult	
32	000000172 000000174	000000173 000000175	Perf3	TgtI	WHuge

NUMBER OF PACKS = 32

CLUSTER SUMMARY REPORT

THRESHOLD DENSITY, K= 2
NUMBER OF COMPONENT PACKS ANALYSED= 32
NUMBER OF GROUPS FORMED= 11

CLUSTER NO. 1

M/CS USED:
Perf1 WSmall TgtIV TgtIII TgtI

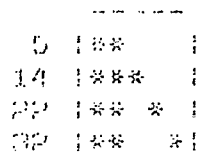
COMPONENT PACK VERSUS MACHINE MATRIX

```
     -----
 4  |**    |
 9  |***   |
12  |** *  |
18  |**   *|
```

CLUSTER NO. 2

M/CS USED:
Perf3 WHuge TgtIV TgtII TgtI

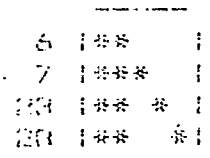
COMPONENT PACK VERSUS MACHINE MATRIX



CLUSTER NO. 3

M/CS USED:
Perf1 WHuge TgtIV TgtII TgtV

COMPONENT PACK VERSUS MACHINE MATRIX



CLUSTER NO. 4

M/CS USED:
Perf1 WMult TgtIV TgtIII

COMPONENT PACK VERSUS MACHINE MATRIX

```
    ----
15 |**   |
16 |***  |
19 |** *|
```

CLUSTER NO. 5

M/CS USED:
Perf2 WHuge TgtIV TgtI TgtII TgtIII

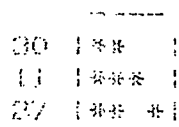
COMPONENT PACK VERSUS MACHINE MATRIX

```
    ------
26 |**     |
21 |***    |
24 |** *·  |
25 |**  *  |
29 |**   *|
```

CLUSTER NO. 6

M/CS USED:
Perf1 WLarge TgtIV TgtIII

COMPONENT PACK VERSUS MACHINE MATRIX



CLUSTER NO. 7

M/CS USED:

Perf3 TgtIV WSmall

COMPONENT PACK VERSUS MACHINE MATRIX

CLUSTER NO. 8

M/CS USED:
Perf2 TgtIV WMult

COMPONENT PACK VERSUS MACHINE MATRIX

```
  --
 8 |***|
11 |*  *|
```

CLUSTER NO. 9

M/CS USED:
Perf2 TgtIV WSmall

COMPONENT PACK VERSUS MACHINE MATRIX

```
  --
10 |***|
 1 |*  *|
```

CLUSTER NO. 10

M/CS USED:
Perf3 TgtIV WMult

COMPONENT PACK VERSUS MACHINE MATRIX

```
   ---
13 |***|
20 |* *|
```

CLUSTER NO. 11

M/CS USED:
Perf1 TgtIV WMedum

COMPONENT PACK VERSUS MACHINE MATRIX

```
   ---
17 |***|
```

MACHINES REQUIRING DUPLICATION:

TgtIV TgtI Perf1 WHuge TgtII TgtIII Perf3 WSmall

Perf2 WMult